Engineer

DOM

dynamischer Objektbaum, den ein Browser aus einer Seite baut

Was es ist

Das DOM, kurz für Document Object Model, ist der Objektbaum, den ein Browser aus einer Seite aufbaut und im Speicher behält, solange sie geöffnet ist. Der HTML-Parser liest das Markup und erzeugt ein Dokumentobjekt: Jedes Element, jeder Textabschnitt und jeder Kommentar wird zum Knoten. Jeder Knoten kennt seinen Elternknoten, seine Kinder und seine Nachbarn. Skripte lesen und ändern die Seite über diesen Baum, und aus ihm wird ermittelt, was der Browser zeichnet.

Die WHATWG pflegt das Modell im DOM Standard. Dessen eigene Zusammenfassung nennt drei plattformunabhängig behandelte Dinge: Knotenbäume, Ereignisse und das Abbrechen laufender Aktivitäten. Der Name ist älter als der Standard. Ursprünglich bezeichnete er eine Schnittstelle für den Zugriff auf ein Dokument und dessen Änderung. Heute sagen die meisten auch zum Baum selbst „das DOM“; beide Bedeutungen sind im Alltag gebräuchlich. Der HTML Standard baut darauf auf: HTML definiert die Elemente und ihr Verhalten. Der DOM Standard definiert, was Knoten, Element, Attribut und Ereignis sind.

Der Baum ist nicht die Quelldatei. Das HTML einer Seite ist über das Netzwerk eingetroffener Text. Das DOM ist das Ergebnis seiner Verarbeitung durch den Parser, der jeden Parserfehler nach den Regeln im HTML Standard behandelt. Danach wird es von jedem seither ausgeführten Skript verändert. Der Parser bewahrt leicht vergessene Details. Der Leerraum zwischen zwei Tags kann zu einem eigenen Textknoten werden. Das erste Kind eines Elements ist deshalb nicht immer sein erstes enthaltenes Element. Das eigene Beispiel des Standards merkt allerdings an, dass nicht jeder Leerraumabschnitt zu einem Textknoten wird.

Knoten gibt es in einigen Arten. An der Wurzel steht das Dokument, darunter Doctype, Elemente, Textknoten und Kommentare. Ein Dokumentfragment ist ein leichter Container außerhalb des Seitenbaums. Dort lässt sich bequem ein Block zusammensetzen und anschließend auf einmal einfügen. Ein Element kann auch einen Schattenbaum beherbergen, einen eigenen getrennten Baum, in den gewöhnliche Dokumentabfragen nicht hineinreichen.

Knotenlisten gibt es in zwei Arten, und der Unterschied zählt. Nach der Regel des Standards ist eine Sammlung dynamisch, sofern nichts anderes festgelegt ist: `childNodes` und die HTMLCollection aus `getElementsByClassName()` ändern sich mit dem Baum. `querySelectorAll()` ist die ausdrücklich genannte Ausnahme. Es liefert eine statische Liste, einen Snapshot zum Aufrufzeitpunkt.

Derselbe Standard definiert den Weg von Ereignissen. Ein Ereignis wird an einem Zielknoten ausgelöst und durchläuft drei Phasen: Capturing auf dem Weg von der Baumspitze durch die Vorfahren des Ziels nach unten, die Phase am Ziel selbst und Bubbling zurück durch die Vorfahren nach oben. Die Eigenschaft `eventPhase` meldet die Phase als 1, 2 oder 3 sowie 0, wenn das Ereignis gerade nicht zugestellt wird. Ein mit `addEventListener()` hinzugefügter Listener nimmt Optionen an: `once` entfernt ihn nach seinem ersten Aufruf, `passive` verspricht, dass er `preventDefault()` nicht aufruft, und `signal` bindet ihn an einen `AbortController`, dessen Abbruch den Listener entfernt.

Das Konzept für eine Entscheidung nutzen

Nutzen Sie „Was ist das DOM?“ für eine konkrete Entscheidung. Wir können helfen, das Konzept mit dem System in Verbindung zu bringen, das Sie entwickeln oder einkaufen. Oberflächen, die schnell laden, auf dem Smartphone funktionieren und sich in allen Browsern Ihrer Kunden gleich verhalten.

Unsere Leistungen entdecken Kontakt aufnehmen →

Die Annahmen hinter dem Begriff prüfen

Nutzen Sie die Erklärung und die verlinkten Arbeiten, um Fragen für Ihr Projekt zu formulieren. Wir können die Annahmen gemeinsam mit Ihnen prüfen, bevor ein Werkzeug ausgewählt wird. Full-Stack-GIS-Kartenentwicklung geleitet — Full-Stack-Kartenentwicklung von der Datenbank bis zum Browser

Full-Stack-GIS-Kartenentwicklung geleitet Kontakt aufnehmen →

Warum es wichtig ist

Jedes Skript, das eine Seite ändert, arbeitet über das DOM, unabhängig von Sprache oder Framework. Ein Framework legt eine eigene Schicht darüber, doch was der Leser am Ende sieht, sind Knoten dieses Baums. Next.js beschreibt Hydration als das Anfügen von Ereignishandlern an das DOM durch React. So wird vom Server gesendetes HTML zu einer auf Klicks reagierenden Seite.

Ein Team, das den darunterliegenden Baum kennt, kann das tatsächliche Verhalten des Frameworks debuggen, statt nur von den ihm erteilten Anweisungen auszugehen.

Im Baum trennen sich auch Text und Markup. Eine `textContent` zugewiesene Zeichenkette ersetzt die Kinder eines Elements durch einen einzelnen Textknoten und wird nie als Markup gelesen. Dieselbe Zeichenkette an `innerHTML` geht an den HTML-Parser. Der HTML Standard merkt an, dass diese Setter standardmäßig nichts Gefährliches entfernen. Die Wahl zwischen beiden entscheidet, ob der Anzeigename eines Besuchers als Text erscheint oder als Code läuft.

Was ein Leser erlebt, hängt von mehr als dem Baum ab. Styles bestimmen, ob ein Knoten sichtbar ist, wo er liegt und ob ein Zeiger ihn erreicht. Ein nur die Knoten durchlaufendes Werkzeug sieht nichts davon. In dieser Lücke irren automatisierte Prüfungen am häufigsten. Deshalb muss die gerenderte Seite ebenso geprüft werden wie ihr zugrunde liegendes Dokument.

Schließlich ist das DOM der Vertrag zwischen einer Seite und allem, was sie nach getaner Serverarbeit liest. Tests, die sich durch eine Seite klicken, Browsererweiterungen und assistive Software beginnen alle beim dynamischen Baum. Die leicht übersehene Gegenseite dieses Vertrags: Ein Crawler ohne JavaScript-Ausführung sieht nur das gesendete HTML und keinen später von Skripten eingefügten Knoten.

Wie es in der Praxis aussieht

Das folgende, für diesen Artikel geschriebene Skript füllt eine Bestellliste aus Daten. Es hält einen einzigen Klicklistener auf der Liste statt einen pro Eintrag und kann ihn später mit einem Aufruf entfernen. `openOrder` steht für die Aktion der Seite bei Auswahl einer Bestellung.

```
const list = document.querySelector("#orders");
const controller = new AbortController();

function render(orders) {
  const items = orders.map((order) => {
    const li = document.createElement("li");
    li.dataset.id = order.id;
    li.textContent = `${order.number}: ${order.status}`;
    return li;
```

```
  });
  list.replaceChildren(...items);
}

list.addEventListener(
  "click",
  (event) => {
    const li = event.target.closest("li");
    if (li) openOrder(li.dataset.id);
  },
  { signal: controller.signal },
);

// When the view goes away:
controller.abort();
```

Jeder Teil entspricht dem Standard. `createElement()` erstellt ein Element ohne Elternknoten, bis es eingefügt wird. `textContent` setzt die Bestellnummer als Text ein. Ein Status mit spitzen Klammern bleibt deshalb ein Status. `replaceChildren()` entfernt die alten Einträge und fügt die neuen in einer Operation ein. Der eine Listener auf der Liste bedient jeden Eintrag, weil ein Klick auf einen Eintrag bis zur Liste hochsteigt. `closest()` geht vom angeklickten Objekt zum nächsten `li`. Der Abbruch des Controllers entfernt den Listener, ohne dass die Seite eine Referenz auf die Funktion behalten muss.

Der Baum erklärt auch eine Fehlerklasse aus dem Review dieser Website. Bei einem Barrierefreiheitsreview im August 2026 bestand die Website ihre automatisierten Regeln und hatte trotzdem elf Defekte. Jeder lag an der gerenderten Seite statt an ihrem Markup. Acht Footerlinks blieben hinter `pointer-events` in der Tabulatorreihenfolge. Diese Eigenschaft entfernt nur Zeigerinteraktion. Für alles, was das Dokument liest, waren es gewöhnliche Links. Ein Zeiger erreichte sie nie, die Tastatur hielt weiterhin an jedem. Eine scrollgesteuerte Animation schnitt den Footer auf drei Seiten ab. Von den vier hinterlassenen Regeln warnt eine davor, nur dem Baum zu vertrauen: Sichtbarkeit muss anhand gerenderter Pixel und nicht nur des Dokumentbaums festgestellt werden.

Wo es schiefgeht

Eine Schleife über eine dynamische Sammlung überspringt jeden zweiten Eintrag. Die HTMLCollection aus `getElementsByClassName()` ist dynamisch. Entfernt eine Schleife per Index die Klasse von jedem Element, verkleinert sie dabei die

Sammlung und überspringt jedes zweite Element. Durchlaufen Sie stattdessen eine statische Liste: `querySelectorAll()` liefert eine, und `Array.from()` macht aus einer dynamischen Sammlung zunächst ein gewöhnliches Array.

```
const pending = document.getElementsByClassName("pending");
for (let i = 0; i < pending.length; i++) {
  pending[i].classList.remove("pending"); // skips every other one
}

for (const el of document.querySelectorAll(".pending")) {
  el.classList.remove("pending"); // static list: all of them
}
```

Listener häufen sich an. Eine View, die bei jedem Rendern Listener hinzufügt und nie entfernt, führt jeden Handler bei einem Ereignis so oft aus, wie die View gerendert wurde. Nach drei Besuchen sendet ein Klick drei Anfragen. Für das manuelle Entfernen braucht es genau dasselbe Funktionsobjekt wie beim Hinzufügen. Eine direkt am Aufruf geschriebene Pfeilfunktion hinterlässt keine Referenz, die sich später übergeben ließe. Übergeben Sie pro View das `signal` eines `AbortController`, wie im Beispiel oben. Ein einziges `abort()` entfernt dann alle von der View hinzugefügten Listener. Nutzen Sie `once: true` für einen Listener, der nur einmal auslösen soll.

Text geht als Markup hinein. Wer aus Daten einen HTML-String baut und ihn `innerHTML` zuweist, übergibt die Daten dem Parser, standardmäßig ohne Bereinigung. Ein Anzeigename mit Markup, etwa einem Bildtag mit Fehlerhandler, landet als echtes Element samt Handler auf der Seite. Text gehört in `textContent` oder einen mit `createTextNode()` erzeugten Textknoten. Muss Markup tatsächlich aus einer Zeichenkette kommen, definiert der HTML Standard `setHTML()`. Es bereinigt beim Parsen und entfernt unsichere Inhalte.

Ein Observer füttert sich selbst. Ein `MutationObserver` meldet Änderungen an beobachteten Knoten. Sein Callback läuft nach jedem Änderungsblock. Schreibt ein Callback in den beobachteten Teilbaum, etwa um einen Zähler hinzuzufügen oder eine Klasse zu korrigieren, erzeugt er neue Einträge und wird erneut aufgerufen. Die Seite kann sich in einer selbst erzeugten Schleife aufhängen. Beobachten Sie nur das Nötige, mit `attributeFilter` und möglichst ohne `subtree`. Lassen Sie den Callback vor dem Schreiben prüfen, ob seine Änderung schon vorliegt, oder rufen Sie vor dem Schreiben `disconnect()` und danach erneut `observe()` auf.

Weiterführende Lektüre

- DOM Standard — die WHATWG-Definition von Knoten, Bäumen, dynamischen und statischen Sammlungen, Ereignissen, `AbortController` und `MutationObserver`.
- HTML-Dokumente parsen — wie der HTML-Parser einen Zeichenstrom in ein Dokumentobjekt umwandelt, einschließlich Parserfehlern.
- Dynamisches Einfügen von Markup — `innerHTML`, `setHTML()` und `setHTMLUnsafe()` und welche davon bereinigen.
- Server- und Clientkomponenten — die Next.js-Erklärung von Hydration: HTML vom Server, danach an das DOM angefügte Ereignishandler.

Glossar

Für Ihre tatsächlichen Rahmenbedingungen auswählen

Beschreiben Sie den Arbeitsablauf, die vorhandenen Schnittstellen und die Nutzer, die die Veränderung benötigen. Wir können eine prüfbare Lieferung definieren und die Übergabe so gestalten, dass Ihr Team die gelieferten Bestandteile warten kann.

Preise ansehen Kontakt aufnehmen →

Die Begriffe praktisch nutzbar machen

Fragen Sie, wie „Was ist das DOM?“ auf Ihre Organisation zutrifft. Ein konkretes Beispiel aus Ihrem bestehenden System reicht aus, um das Gespräch zu beginnen.

Unsere Leistungen entdecken

welcome@engineer.company +45 50 10 61 84 Kopenhagen, Dänemark

Nehmen Sie diese Seite mit

Kontaktkarte

Mit Sorgfalt und Liebe gestaltet von D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Copyright © 2025 – heute · Engineer ApS · Alle Rechte vorbehalten.

Gedruckt von https://engineer.company/de/glossary/dom/